QubesOS articles

GPU

Author:
Neowutran

Contents

1. HARDWARE

To have an 'HVM' for gaming, you must have

- A dedicated GPU. By dedicated, it means: it is a secondary GPU, not the GPU used to display dom0. In 2023, 'Nvidia' and 'Amd' GPU work. Not tested with Intel GPUs. (External GPU using thunderbolt work)
- A lot of patience. GPU passthrough is not trivial, and you will need to spend time debugging.
- A screen available for the gaming 'HVM'. (It can be a physical monitor or just to have multiple cables connected to the screen and switching between input source) **[Optional]**
- Dedicated gaming mouse and keyboard **[Optional]** .

2. IOMMU GROUP

2.1 Goal

What The goal of this step if to retrieve the default IOMMU Group (VFIO - "Virtual Function I/O" - The Linux Kernel documentation) of your hardware.

Why It can help understanding potential issue with your setup (what devices live in the same IOMMU group as your GPU) / finding potential workaround. If you feel lucky, skip this step.

How You can't see your IOMMU Group when you are using Xen (the information is hidden from dom0).

- Boot a live linux distribution
- In the grub, enable iommu: Add the parameters `iommu=1 iommu_amd=on` to the linux commandline
- Once you logged in your live linux distribution, you need to retrieve the folder structure of `/sys/kernel/iommu_group`. You can use the following script to do that:

```
#!/bin/bash
shopt -s nullglob
for g in /sys/kernel/iommu_groups/*; do
 echo "IOMMU Group ${g##*/}:"
 for d in $g/devices/*; do
  echo -e "\t$(lspci -nns ${d##*/})"
done
done
```

3. GRUB MODIFICATION

You must hide your secondary GPU from dom0. To do that, you have to modify the GRUB. In a dom0 Terminal, type:

```
qvm-pci
```

Then find the devices id for your secondary GPU. In my case, it is `dom0:0a_00.0`. Edit /etc/default/grub, and add the PCI hiding.

```
GRUB_CMDLINE_LINUX="... rd.qubes.hide_pci=0a:00.0 "
```

then regenerate the grub

```
grub2-mkconfig -o /boot/grub2/grub.cfg
```

If you are using UEFI and Qubes OS 4.2 or earlier, the file to override with `grub2-mkconfig` is `/boot/efi/EFI/qubes/grub.cfg`.

Note: if after this step when you reboot the computer you get stuck in the QubesOS startup that means you are trying to use the GPU you just hid. Check your BIOS options. Also check the cables, BIOS have some GPU priority based on the type of cable. For example, DisplayPort can be favoured over HDMI.

Once you have rebooted, in dom0, type `sudo lspci -vvn`, you should see "Kernel driver in use: pciback" for the GPU you just hid.

4. PREPARING THE GUEST

As of 2025, I recommend using a Linux guest instead of a window guest.

4.1 Windows

Install a window VM, you can use this qvm-create-windows-qube

4.2 Linux

Create a new standalone Qube based on the template of your choice.

You must run the kernel provided by the guest distribution, because we will use some non-default kernel module for the GPU driver. Just follow the doc: managing-vm-kernel.

Install the GPU drivers you need.

5. PASS THE GPU

In qubes settings for the HVM, go to the 'devices' tab, pass the ID corresponding to your GPU.

```
qvm-pci attach gpu_gaming_archlinux dom0:0a_00.0 --persistent
```

You may or may not need to add the option "permissive" or "no-strict-reset". You may or may not need to passthrough additional devices depending on the result of the IOMMU script.

Some word about the security implication of thoses parameters.

```
qvm-pci attach gpu_gaming_archlinux dom0:0a_00.0 -o
↪ permissive=True -o no-strict-reset=True --persistent
```

6. STARTING THE GUEST

This is where you will have a lot of issues to debug. Prepare for intense suffering

For Linux guests, run 'sudo dmesg' to have all the kernel log indicating you if there is a issue with your GPU driver. For some hardware, the MSI calls won't work. You can work around that using for example `pci=nomsi` or `NVreg_EnableMSI=0` or something else. Check your drivers options. Check if alternative drivers exist (amdgpu, nvidia, nouveau, nvidia-open, using drivers from the official website, ...). Check multiple kernel version.

For nvidia GPU, I recommand using "nvidia-open" drivers instead of "nvidia"

Some links that could help you to debug the issues you will have

- https://forum.qubes-os.org/t/ryzen-7000-serie/
- https://dri.freedesktop.org/docs/drm/gpu/amdgpu.html

For windows guests you will probably have the same issues but it will be harder to debug. I recommend using the drivers from Windows Update instead of the official drivers from the website of the constructor.

Some things that may be useful for debugging:

- Virsh (start, define, ...)
- /etc/libvirt/libxl/
- xl
- /etc/qubes/templates/libvirt/xen/by-name/
- /usr/lib/xen/boot/
- virsh -c xen:/// domxml-to-native xen-xm /etc/libvirt/libxl/...

Issues with the drivers could be related to 'qubes-vmm-xen-stubdom-linux', 'qubes-vmm-xen', and the Linux kernel you will be using.

7. [CASE: NO DEDICATED SCREEN, MOUSE OR KEYBOARD] VIRTUALGL AND VULKAN

In some cases, it will just work out of the box (application will automatically detect the GPU and will be able to use it without modification.

In some cases, an additionnal layer is required: using VirtualGL, mentionned here: Seamless GPU passthrough on Qubes OS with VirtualGL

In that case you may need to specify a new variable

```
LD_LIBRARY_PATH=/usr/lib
```

For nvidia GPU owner, with a hardware that support "Nvidia Prime Offload", Prime offload can be activated with environment variable:

```
__NV_PRIME_RENDER_OFFLOAD=1 __VK_LAYER_NV_optimus=NVIDIA_only
↪  __GLX_VENDOR_LIBRARY_NAME=nvidia
```

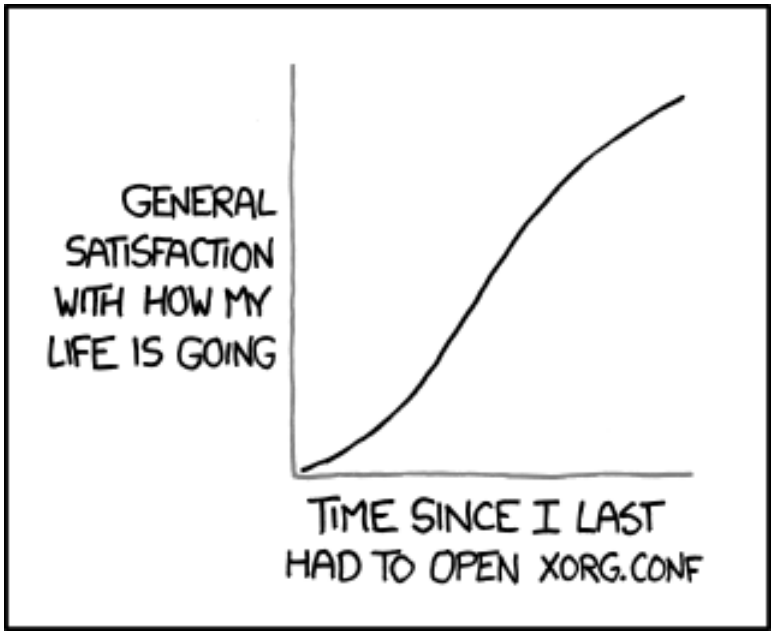
QubesOS forum - Atrate configuration

You may have issue with how the mouse is handled by the game. AFAIK, no one tried to solve this issue.

8. [CASE: DEDICATED SCREEN, MOUSE AND KEYBOARD] LINUX GUEST — INTEGRATION WITH QUBESOS

8.1 Xorg

Now Xorg. From XKCD:



Things you need to install:

- The Xorg input driver to support your mouse and keyboard
- Your favorite Windows Manager

In my case, it is:

archlinux version:

```
pacman -S xorg i3
```

debian version:

```
apt install xserver-xorg-input-kbd xserver-xorg-input-libinput
↪  xserver-xorg-input-mouse i3
```

Then create a XORG configuration file for your GPU and screen. My file named 'xorg.conf':

```
Section "ServerLayout"
Identifier "Passthrough Layout"
Screen 0 "Passthrough Screen" Absolute 0 0
EndSection

Section "Device"
Identifier  "Passthrough GPU"
# name of the driver to use. Can be "amdgpu", "nvidia", or
 ↪  something else
Driver      "driver"
Option "Coolbits" "4"
# The BusID value will change after each qube reboot.
BusID       "PCI:0::0"
EndSection

Section "Monitor"
Identifier "Passthrough monitor"
EndSection

Section "Screen"
Identifier "Passthrough Screen"
Device     "Passthrough GPU"
Monitor    "Passthrough Monitor"
EndSection
```

We can't know what is the correct BusID before the qube is started. And it change after each reboot. So let's write a script — named "xorgX1.sh" — that update this configuration file with the correct value, then start a binary on the Xorg X screen n°1.

```
#!/bin/bash
binary=${1:?binary required}

# Find the correct BusID of the AMD GPU, then set it in the
↪  Xorg configuration file
lspci | grep "VGA" | grep -E "NVIDIA" && sed -i 's/^Driver
↪  .*/Driver "nvidia"/g' /opt/xorg.conf
lspci | grep "VGA" | grep -E "AMD/ATI" && sed -i 's/^Driver
↪  .*/Driver "amdgpu"/g' /opt/xorg.conf
pci=$(lspci | grep "VGA" | grep -E "NVIDIA|AMD/ATI" | cut -d "
↪  " -f 1 | cut -d ":" -f 2 | cut -d "." -f 1 | cut -d "0" -f
↪  2)
sed -i 's/"PCI:[^"]*"/"PCI:0:'$pci':0"/g' /opt/xorg.conf

# Start the Xorg server for the X screen number 1.
# The X screen n__0 is already used for QubesOS integration
sudo startx "$binary" -- :1 -config /opt/xorg.conf
```

8.2 Audio

- Create a script to launch your favorite Windows Manager and force it to use a specific pulse server. Example “i3.sh”:

```
#!/bin/bash
sleep 5 && sudo setxkbmap -display :1 fr &
/bin/sudo -u user PULSE_SERVER=unix:/run/user/1000/pulse/native
↪  bash -c 'sudo xhost + local:;/usr/bin/i3'
```

And launch it:

```
sudo ./xorgX1.sh ./i3.sh
```

9. AUTOMATING MOST OF THAT

I tried to write a script to automate all the previous steps. It is available here: gpu_template

Please be carefull and read the code, not much tests have been done

9.1 Issues and fixes

In one case in a setup with Intel IGPU + Nvidia DGPU, dom0 xorg crashed. Solved the case by adding a Xorg configuration to explicitly use Intel:

GPU Passthrough - lightdm.service won't start

9.2 References

- Archlinux: PulseAudio
- Archlinux: PulseAudio/Troubleshooting

9.3 Contributors

- @deeplow
- @xvrhthxn
- neowutran